1. Floating Point

1. Watch the following video: Fast Inverse Square Root – A Quake III Algorithm `https://youtu.be/p8u_k2LIZyo`.

   Where does the constant `0x5f3759df` come from?

   (Just a quick answer, otherwise watching the video isn't much of a question. But watching the video is the main bit of work, not writing down this. You can watch the video on 2× speed if you can still follow it. And you don't have to know the fast inverse square root for the exam.)

   [medium, because of video length]

2. JavaScript uses IEEE 754 double-precision floating point (1 bit of sign, 11 bits of exponent, and 52 bits of mantissa [remember the implicit `1.` so it is 53 bits of significand]). What is the range of integers which can be represented contiguously (without gaps)?

   [small]

3. Solve exercise 1.6 of the lecture notes (copied for convenience).

   Another example of the inaccuracy of floating-point arithmetic takes the golden ration $\varphi \approx 1.618$ as its starting point:

   $$\gamma_0 = \frac{1+\sqrt{5}}{2} \quad \text{and} \quad \gamma_{n+1} = \frac{1}{\gamma_n - 1}$$

   In theory it is easy to prove that $\gamma_n = \ldots = \gamma_1 = \gamma_0$ for all $n > 0$. Code this computation in OCaml and report the value of γ_{50}. *Hint:* in OCaml, `sqrt 5` is expressed as `sqrt 5.0`.

   [small]

2. Complexity

1. Solve one of the following equations:

   $$T(1) = 1 \qquad T(n) = 2T\left(\frac{n}{2}\right) + 1$$

   or

   $$T(1) = 1 \qquad T(n) = T\left(\frac{n}{2}\right) + n$$

   [medium]

2. The Fibonacci function can be written as

   ```OCaml
   let rec fib(n) =
       if n<2 then 1
       else fib(n-2) + fib(n-1) ;;
   ```

   What is its time and space complexity?

   [small]

3. The Fibonacci function can be more efficiently written if instead of returning a single value of `fib(n)`, we return two values, `(fib(n), fib(n-1))`. Write this function.

   What is its time and space complexity?

   [small]

3. Lists and recursion

1. What is the difference between `List.fold_left` and `List.fold_right`? Write your own version of them (say `foldl` and `foldr`) using pattern-matching on the list.

   [small]

2. List folding is the principal function for operating on lists, all other functionality can be implemented on top of `fold_left`. Using `fold_left`, implement `rev l` for reversing the list `l`, and `map f l` for applying `f` to each element of the list `l`.

   [small]

3. Write a function which takes a list of strings and formats it, for example `fmtlist [1;2;3]` evaluates to `"[1; 2; 3]"`. You might want to write this function in terms of `foldl` to help with the next part. Note, string concatenation is done using the `^` operator in OCaml.

   [small]

4. Fold-left is such a fundamental operation of lists, that lists can be encoded as a function that represents fold-left. (Can be encoded, but not encoded this way usually.) For example,

```OCaml
let l     (* encoding [] *)          = fun f v -> v                ;;
let l_1   (* encoding ["1"] *)       = fun f v -> f v "1"          ;;
let l_1_2 (* encoding ["1"; "2"] *) = fun f v -> f (f v "1") "2" ;;

let foldl f v l = l f v ;;

(* Copy/re-evaluate your `fmtlist` to use the new fold *)
fmtlist l     ;; (* Evaluates to "[]" *)
fmtlist l_1   ;; (* Evaluates to "[1]" *)
fmtlist l_1_2 ;; (* Evaluates to "[1; 2]" *)
```

   Can you come up with a function that conses an element to a list, in this representation? So that

```OCaml
fmtlist (cons "0" l_1_2) ;; (* Evaluates to "[0; 1; 2]" *)
```

   [big, if you can't figure it out it's okay]